XML Filtering for Publish/Subscribe System in a Mobile Environment

Yi Yi Myint, Dr.Hnin Aye Thant
University of Technology (Yatanarpon Cyber City)
yiyimyint.utycc@gmail.com,hninayethant@gmail.com

Abstract

The publish/subscribe model has gained acceptance as a solution for the loose coupling of systems in terms of asynchronous communication enabling Selective Dissemination of Information (SDI) to mobile clients. The advent of Extensible Markup Language (XML) as a de facto standard for information exchange and the development of query languages for XML data enable the development of more sophisticated filtering mechanisms. The inherent limitations of mobile devices necessitate information to be delivered to mobile clients to be highly personalized according to user profiles. This paper proposes an approach that integrates publish/subscribe system and XML message filtering by describing indexing mechanism to enhance Xfilter algorithm based on a modified Finite State Machine (FSM) approach that can quickly locate and evaluate relevant profiles.

1. Introduction

The number of applications using XML data representation is growing rapidly, thus the process of XML filtering is becoming an essential need of different application areas such as publish/subscribe (pub/sub) system, peer-to-peer networks and web services. Publish/subscribe systems [4] grow rapidly, targeting many areas such as news monitoring, ecommerce site monitoring and alerting services for digital libraries.

XML pub/sub has emerged as a solution for loose coupling of disparate systems at both the communication and content levels. At the communication level, pub/sub enables loose coupling of senders and receivers based on the receivers' data interests. With respect to content, XML can be used to encode data in a generic format that senders and receivers agree upon due to its flexible, extensible, and self-describing nature; this way, senders and receivers can exchange data without knowing the data representation in individual systems. Therefore, the main purpose of an XML filtering system is to find all the user profiles that have a match with a specific XML document.

In pub/sub systems, a subscriber registers a subscription that are of interest to them, called profiles to the pub/sub service and receives published messages that match the subscription. Intuitively, the source (publishers) can allow their subscribers to retain whatever they want, and send all the data to all subscribers. The first is to find the subscriber by using the subscription information, and then send appropriate data to subscribers [8]. Data matching can be performed either at the source or at some centralized brokers. Several approaches for XML filtering have been reported in the literature.

Expressing highly personalized profiles need a querying power just like SQL provides on relational databases. Since the queries will be executed on the documents fetched over the Internet, it is natural to expect the documents to be in XML [3]. XML being the emerging standard for data exchange over the Internet. Then the user profiles need to be defined through an XML query language. XML-QL is a good candidate in this respect due to its expressive power as well as its elaborate mechanisms for specifying query results through the CONSTRUCT statement.

The rest of the paper is organized as follows: Section 2 briefly summarizes the related work. Section 3 discusses the issues of XFilter to be used in mobile environment. In Section 4, the overall architecture of the system is described. The operation of the system that includes creating query index, operation of the finite state machine, proposed filtering algorithm and generating customized results are explained in Section 5. Section 6 gives the expected performance of the system. Finally Section 7 concludes the paper.

2. Related Work

The filtering mechanism described in this paper is influenced by the XFilter system [1]. XFilter is designed and implemented for pushing XML documents to users according to their profiles expressed in XML Path Language (XPath). It takes the advantage of embedded schema information in the XML documents to create better user profiles compared to existing keyword based systems. While doing that, it provides efficient filtering of XML documents with the help of profile index structures in its filter engine. XFilter converts each XPath query into a Finite State Machine (FSM) to deal with XPath structures effectively. However as the name implies, Xfilter is a filtering mechanism; it does not execute the XPath queries to produce results. Therefore when a document matches a user's profile, the whole document is pushed to the user. This requires lots of storage capacity that a mobile device would not be able to handle.

YFilter [2] overcomes the disadvantage of XFilter by using nondeterministic finite automata (NFA) to emphasize prefix sharing. However, the ancestor/descendant relationship introduces more

matching states, which may result in the number of active states increasing exponentially. Postprocessing is required for YFilter. To deal with queries with nested paths (complex queries), YFilter decomposes them into simple queries and matches them separately.

BFilter [7] conducted the XML message filtering and matching by leveraging branch points in both the XML document and user query. It evaluates user queries that use backward matching branch points to delay further matching processes until branch points match in the XML document and user query. In this way, XML message filtering can be performed more efficiently as the probability of mismatching is reduced. A number of experiments have been conducted and the results demonstrate that BFilter has better performance than the well-known YFilter for complex queries.

XFIS [9] proposed an efficient technique for matching user profiles that is based on the use of holistic twig-matching algorithms and was more effective, in terms of time and space complexities, in comparison with previous techniques. The proposed algorithm was able to handle order matching of user profiles, while its main positive aspect was the envisaging of a representation based on Prüfer sequences that permits the effective investigation of node relationships.

3. Issues of XFilter in Mobile Environment

The XFilter for Mobile clients need to develop to address several shortcomings in XFilter. One of these was that on matching, the whole document is returned to the user. This feature prevents XFilter to be used in mobile environments since the limited capacity of the mobile devices is not enough to handle or process the entire document let alone to receive it. Furthermore, XFilter does not exploit the commonalities between the queries, i.e. it generates a FSM per query. This observation motivated us to develop mechanisms that use only a single FSM for the queries which have common element structure.

Another point is that in case the user profiles are complex, a more powerful language than XPath is needed, and in this case the choice was XML-QL. Not only is XML-QL more powerful than XPath, it is considered the most powerful among all XML query languages. In providing customized results to the mobile clients, the result construction features of XML-QL also help. When such a system providing highly personalized services is deployed on the Internet, the performance becomes a critical issue since the number of users can easily grow dramatically. A key challenge is then to efficiently and quickly process the potentially huge set user profiles on XML resources. This boils down to developing efficient ways of processing XML-QL queries on XML documents.

4. Overall Architecture of the System

This system proposes the architecture for mobile network to deliver highly personalized information from XML resources to mobile clients whenever the query is satisfied by incoming XML documents. There are two important operations performed in a pub/sub system: XML message filtering and multicast. This system focuses on techniques for XML message filtering. The overall architecture of the system is depicted in Figure 1.

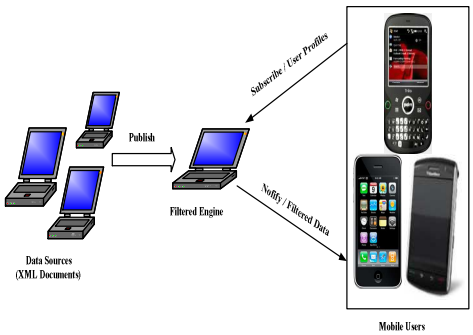


Figure 1. Overall Architecture of the System

Data Sources contains the XML document files that publish messages to the filtered engine component of the pub/sub system. Mobile users are also called subscribers that register a subscription providing the graphical user interfaces to define their profiles from their mobile phones. These profiles are converted into XML-QL queries. The queries can be change- based. It means they need to be activated when the related XML documents change that causes the related FSMs to change their states. The queries are grouped and indexed such that each element in a query group corresponds to a state in the Finite State Machine (FSM).

Filtered engine first parses and creates query indices for user profiles, and also parses the incoming XML documents to obtain the query results. When XML documents and user profiles match, the matched data are stored in a special content list, so that the whole document need not be sent. The filtered engine notifies and sends filtered XML documents to the related mobile clients. Extracting parts of an XML document can save bandwidth in a mobile environment.

4.1. Profile Language using XML-QL

XML-QL [10] is one of the candidates being considered by the World Wide Web Consortium (W3) for the standard way of querying XML documents. XML-QL has a SELECTWHERE construct, like SQL, that can express queries, to extract pieces of data from XML documents. It can also specify transformations that, for example, can map XML data between Document Type Definitions (DTDs) and can integrate XML data from different sources. Although XML-QL shares some functionality with XML's style sheet mechanism, it supports more data-intensive operations, such as joins and aggregates, and has better support for

constructing new XML data or specifying query results through the CONSTRUCT statement, which is required by transformations. XML-QL is implemented as a prototype and is freely available in a Java version.

XML-QL has very elaborate mechanisms for specifying query results through the CONSTRUCT statement. The design features of XML-QL are 1) it is declarative, like SQL; 2) it is relational complete, e.g. it can express joins; 3) it can be implemented with known database techniques; 4) it can extract data from existing XML documents and construct new XML documents. XML-QL can express queries as well as transformations, for example, can map XML data between DTDs and can integrate XML data from different sources. A point to be noted here is that the users should not be expected to express their profiles through XML-QL but rather a user-friendly interface should be provided to them to automatically create the XML-QL statements.

```
<Profile>
    <address>…</address>
    <XML-QL>
      WHERE <course>
      <major>ICT</>
      <program>First Year</>
      <syllabus>$n</>
      </> IN "course.xml"
   CONSTRUCT<result><syllabus>$n</></>
    </XML-QL>
    <PushMode><Every>
            <PeriodSize>…</PeriodSize>
            <PeriodType>…</PeriodType>
        </Every>
        <PushTo>
            <address>…</address>
        </PushTo>
  </PushMode>
</Profile>
```

Figure 2. Profile Syntax represented in XML containing XML-QL query

Profiles defined through a graphical user interface are transformed into XML documents which contain XML-QL queries to provide user friendliness and expressive power as depicted in Figure 2.

4.2. Filtered Engine

The basic components of the filtered engine are 1) **an event-based XML parser** which is implemented using Simple API for XML (SAX) for XML documents; 2) **a profile parser** that has an XML-QL parser for user profiles and creates the Query Index; 3) **a query execution engine** which contains the Query Index which is associated with Finite State Machines to query the XML documents; 4) **a delivery component** which pushes the results to the related mobile clients (see Figure 3). When a document arrives at the Filtered Engine, it is run through an XML Parser, and then drives the process of query execution through the query index.

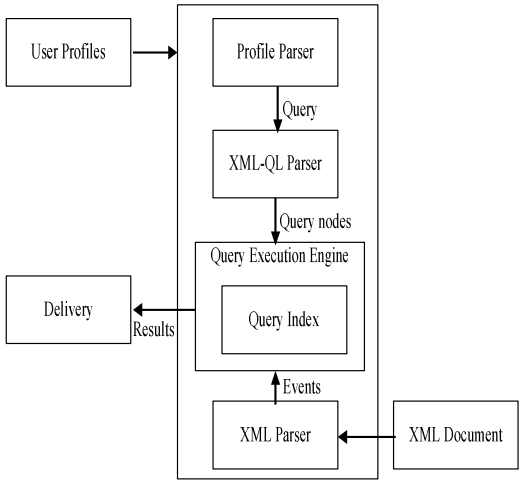


Figure 3. Filtered Engine

5. Operation of the System

The system operates as follows: subscriber informs Filtered Engine when a new profile is created or updated; the profiles are stored in an XML file that contains XML-QL queries, execution conditions (change-base), and addresses to dispatch the results (see Figure 2). The Profile Parser component of the Filtered Engine parses the profiles; XML-QL queries in the profile are parsed by an XML-QL parser. While parsing the queries, the XML-QL parser creates FSM representation of each query, if the query does not match to any existing query group. Otherwise, the FSM of the corresponding query group is used for the input query. FSM representation contains state nodes for each element name in the queries which are stored in the Query Index.

When a new document arrives, the system alerts the Filtered Engine so that the related XML document is parsed. The event based XML parser sends the events encountered to the Query Execution Engine. The handlers in the Query Execution Engine respond to these events. The handlers move the FSMs to their next states when current states succeed certain checks like evaluating the attributes, level checking or pattern matching for character data. In the mean time the data in the document that matches the variables are kept in content lists so that when the FSM reaches its final state, all the necessary partial data to produce the results are there to be formatted and pushed to the related mobile clients.

By enhancing XFilter with FSM, this system is intended to develop to handle very large number of queries and it is quite probable that there will be queries that have the same tree structure and the same element names, that is, the same FSM representation but different constant values. In this case a single FSM can handle these queries and this greatly enhances the performance of the system.

5.1. Creating Query Index

Consider an example XML document and its DTD given in Figure 4.

```
<!-- DTD for Course -->
<!ELEMENT root (course*)>
<!ELEMENT course (degree, major*)>
<!ELEMENT degree (#PCDATA)>
<!ELEMENT major(name, program, semester, syllabus*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT program (#PCDATA)>
<!ELEMENT semester (#PCDATA)>
<!ELEMENT syllabus (sub-code, sub-title, instructor)>
<!ELEMENT sub-code (#PCDATA)>
<!ELEMENT sub-title (#PCDATA)>
<!ELEMENT instructor (#PCDATA)>
<root>
<course>
   <degree>Bachelor</degree>
   <major>
        <name>ICT</name>
        <program>First Year</program>
        <semester>First Semester</semester>
        <syllabus>
            <sub-code>001</sub-code>
            <sub-title>English</sub-title>
            <instructor>Dr. Thiri</instructor>
        </syllabus>
   </major>
</course>…</root>
```

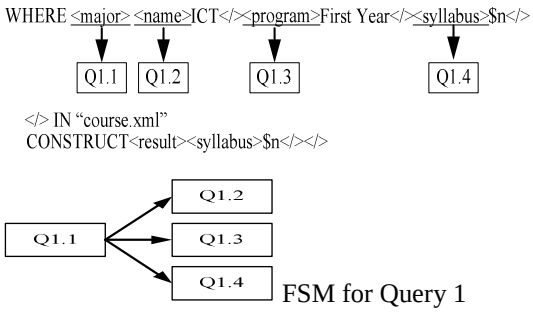
Figure 4. An Example XML Document and its DTD (course.xml)

The example queries and their FSM representations are shown in Figure 5. Note that there is a node in the FSM representation corresponding to each element in the query and the FSM representation's tree structure follows from XML-QL query structure.

The state changes of a FSM are handled through the two lists associated with each node in the Query Index (See Figure 6): The current nodes of each query are placed on the Candidate List (CL) of the index entry for its corresponding element name. All of the query nodes representing future states are stored in the Wait Lists (WL) of their corresponding element name. Copying a query node from WL to the CL represents a state transition in the FSM. Notice that the node copied to the CL also remains in the WL so that it can be reused by the FSM in future executions of the query since the same element name may reappear in another level in the XML document.

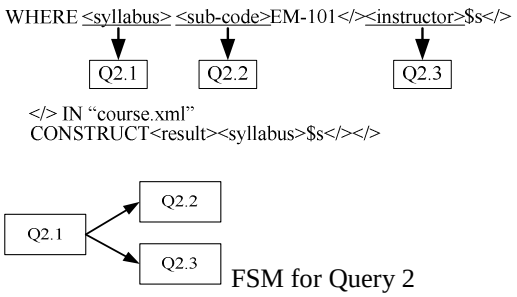
When the query index is initialized, the first node of each query tree is placed on the CL of the index entry for its respective element name. The remaining elements in the query tree are placed in respective WLs. Query nodes in the CL indicate that the state of the query might change when the XML parser processes the respective elements of these nodes. When the XML parser catches a start element tag and if a node in the CL of the element

Query 1: Retrieve all syllabuses of first year program for ICT major.



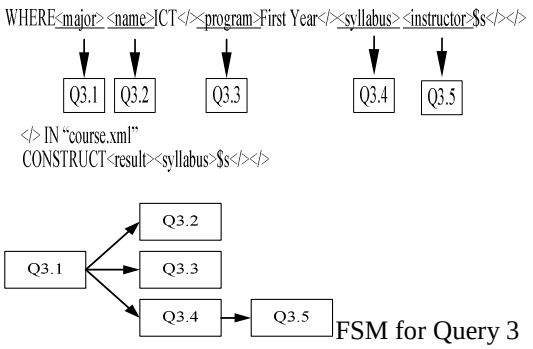
FSM for Query 1

Query 2: Find the instructor name of the subject code EM-101.



FSM for Query 2

Query 3: Retrieve all the instructors in first year program for ICT major.



FSM for Query 3

Figure 5. Example Queries and its FSM Representation

in the Query Index satisfies level check and attribute check, and then the nodes of the immediate child elements of this node in the Query Index are copied from WL to CL. The purpose of the level check is to make sure that the element appears in the document at a level that matches the level expected by the query. The attribute check applies any simple expressions that reference the attributes of the element.

Start Element Handler checks whether the query element matches the element in the document. For this purpose it performs a level and an attribute check. If these are satisfied, depending on the type of the query node it either enables data comparison or starts variable content generation. As the next step, the nodes in the WL that are the immediate successors of this node are moved to CL at this stage. Even in a single document,

the FSM may be executed more than once if the same element names reappear in the document. Therefore there is a need to reinitialize the FSM. Furthermore XML documents can be nested, that is, the same element may appear at different levels. Therefore it may be necessary to generate a FSM to handle this recursion. This is achieved by copying this new node to CL in the query index.

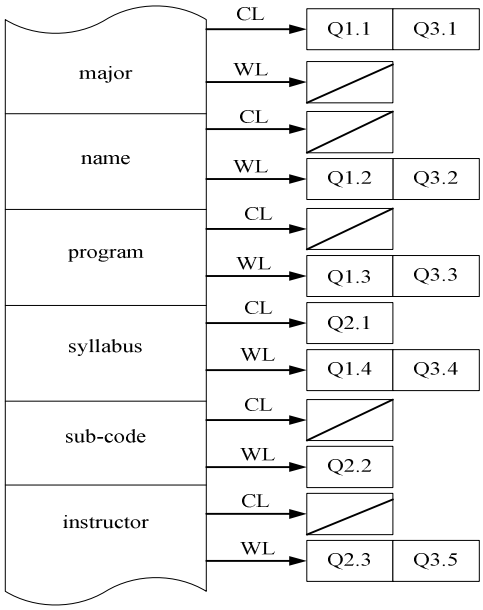


Figure 6. Initial States of the Query Index for Example Queries

End Element Handler evaluates the state of a node by considering the states of its successor nodes and when the root node is reached it generates the output. End element handler also deletes the nodes from CL which are inserted in the start element handler of the node. This provides "backtracking" in the FSM.

Element Data Handler is implemented for data comparison in the query. If the expression is true, the state of the node is set to true and this value is used by the End Element Handler of the current element node.

End Document Handler signals the end of result generation and passes the results to the Delivery Component.

5.3. Proposed Filtering Algorithm

Once the query index has been set up, the algorithm proceeds by reading in a start-element and thus calling the 'Start Element Handler'. The element name in query is looked up in the index and if it's found, then all the nodes in the CL are examined. For each node, a level check is performed to check the level of the node in CL and the matching process is carried out as described in section 5.1. Figure 7 presents the filtering algorithm.

Filtering Algorithm
Input:
QueryIndex qIndex
Incoming element e
List CurrentQueries Q
Init:
qIndex and Q is populated by user requests
While e is not the end of document
If e is in index then
If node level= -1 || node level=element level then
Node ok;
If final node in query then
Query match;
Else if node level ≠ -1 then
Update its level;
End if;
End if;
End if;
End while;
End Algorithm;

Figure 7. Filtering Algorithm

5.4. Generating Customized Results

Results are generated when the end element of the root node of the query is encountered. Content lists of the variable nodes are traversed to fetch content groups. These content groups are further processed to generate results. This process is repeated until the end of the document is reached. The results need to be formatted as defined in the CONSTRUCT clause. The results of the queries that will be sent to related mobile phones.

6. Expected Performance of the System

The proposed architecture is intended to have highly scalable and a very important factor on the performance is the number of query groups and that generating a single FSM per query group rather than per query is well justified. We anticipate that when you have very large number of queries on the same XML document, the probability of having queries with the same FSM representation increases considerably.

As parts of our ongoing work, we are going to show several performance studies such as efficiency, scalability and filtering time by varying 1) number of profiles; 2) depth of queries; 3) number of documents, how much approximate values and exact values will be outputs that our proposed method.

7. Conclusions

As the Web is growing continuously, a great amount of data becomes available to users, making it more difficult for them to discover interesting information by searching. As a result, publish/subscribe systems have emerged in recent years as a promising paradigm. In this paper, we described the architecture for an XML-based publish/subscribe system built on top of a mobile ad hoc

network. We propose indexing mechanism and matching algorithm based on a modified Finite State Machine (FSM) approach that can quickly locate and evaluate relevant profiles. A querying power is necessary for expressing highly personalized user profiles and for the system to be of use to millions of mobile users, it has to be scalable. To achieve high scalability in this architecture, we index the user profiles rather than the documents because of the excessively large number of profiles expected in the system.

References

[1] Altinel, M. and Franklin, M. "Efficient Filtering of XML Documents for Selective Dissemination of Information," Proc of the Int'l Conf on VLDB, Sept 2000. pp. 53-64.

[2] Diao, Y., Altinel, M., Franklin, M., Zhang, H. and Fischer, P. M. "Path Sharing and Predicate Evaluation for High-Performance XML Filtering," ACM Trans. Database Syst., 28(4), Dec 2003, pp. 467–516.

[3] Extensible Markup Language, http://www.w3.org/XML/.

[4] I.Miliaraki, *Distributed Filtering and Dissemination of XML Data in Peer-to-Peer Systems*, PhD Thesis, Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, July 2011.

[5] I.Miliaraki and M. Koubarakis, "Distributed Structural and Value XML Filtering", 4th ACM International Conference on Distributed Event-Based Systems (DEBS 2010), Cambridge, United Kingdom, 2010.

[6] J. Chen, D. DeWitt, F. Tian, Y. Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases", ACM SIGMOD, Texas, USA, June 2000, pp.379-390.

[7] L. Dai, C. Lung and S. Majumdar, "A XML Message Filtering and Matching Approach in Publish/Subscribe Systems", publication in the IEEE Globecom 2010 proceedings.

[8] L. Dai, *XML "Message Filtering and Matching in Publish/Subscribe Systems*," Master Thesis, School of Computer Science, Carleton University, Ottawa, Ontario, Canada, Sept. 2009.

[9] P. Antonellis and C. Makris, "XFIS: an XML filtering system based on string representation and matching", *Int. J. of Web Engineering and Technology*, Vol. 4, Nr 1, 2008.

[10] XML-QL: A Query Language for XML, http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.